APMA 1940X: Topics in Information and Coding Theory
Project 4

Albert Caputo

May 2 2018

1 Random Ribbons via the Medial Axis

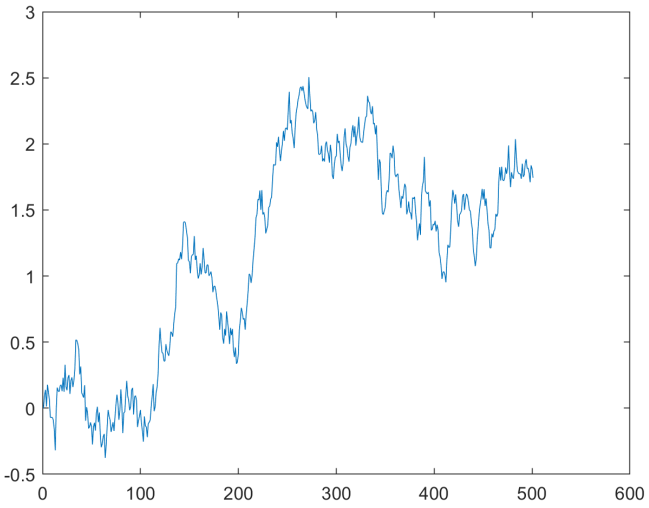
We consider a technique to generate random shapes. Start with a random curve $\Gamma = \text{locus}\{\vec{\gamma}(s)\}$, which we use as the axis, creating a random radius function $r(s)$ and defining the shape as the union of the disks centered on Γ with radius r. To do this, we first generate two Ornstein-Uhlenbeck processes $\mathcal{X}_1, \mathcal{X}_2$ such that

$$\frac{\mathcal{X}_k(t)}{dt} = \theta(\mu - \mathcal{X}_k(t))dt + \sigma dW_t$$

for $k = 1, 2$, $\theta, \mu, \sigma > 0$, and W_t as a standard Brownian motion process. To solve for this computationally, we can discretize over a small $\Delta t > 0$ such that

$$X_k(t + \Delta t) = X_k(t) + \theta(\mu - \mathcal{X}_k(t))\Delta t + \sigma\sqrt{\Delta t}N$$

where $N \sim \mathcal{N}(0, 1)$ is sampled randomly at every time step. To get an idea at what this looks like, we plot a single simulation of $\mathcal{X}$:



For simplicity, we will choose $X_k(0) = 0$. We then solve the following two ODEs to obtain $r(t)$ and $\mathcal{K}(t)$ (the curvature of $\gamma(t)$):

$$\frac{1 - r''(t)r(t) - r'(t)^2}{\sqrt{1 - r'(t)^2}} = \frac{e^{\mathcal{X}_1} + e^{\mathcal{X}_2}}{2}$$

$$r(t)\mathcal{K}(t) = \frac{e^{\mathcal{X}_1} - e^{\mathcal{X}_2}}{2}$$

To solve for $r(t)$, we use MATLAB's ODE integrator ode45 until $|r'(t)| = 1$, which marks the endpoint of the shape. Finally, we solve for $\gamma(t)$ using the ODE

$$\gamma''(t) = \mathcal{K}(t)\gamma'(t)^{\perp}$$

with MATLAB's ode45. Once we have the necessary values, we can determine $P_{\pm}(t)$ as the top and bottom boundary of our random shape up to translation and rotation as follows:

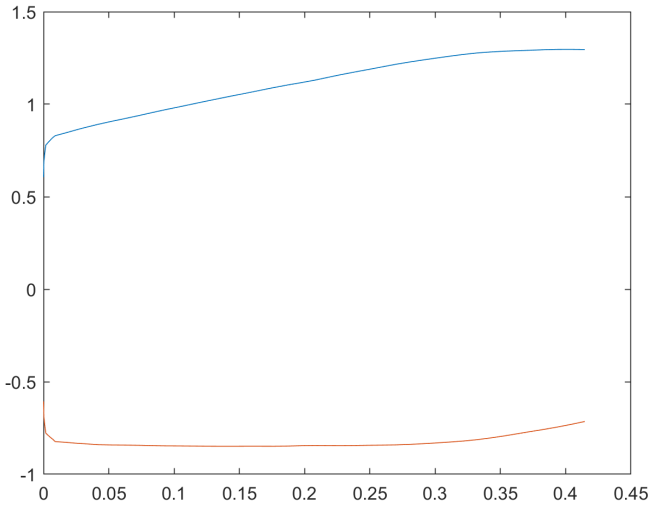
$$P_{\pm}(t) = \gamma(t) + r(t)\big(-r'(t)\gamma'(t) \pm \sqrt{1 - r'(t)^2}\gamma'(t)^{\perp}\big).$$

Once we have P, we can plot our results. Our random shapes will be very dependent upon the choice of the parameters

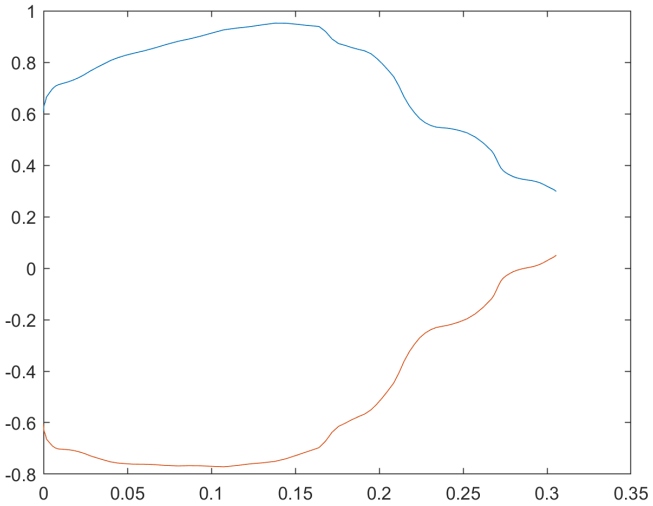
$$\mu, \sigma, \theta, r(0), r'(0), \gamma(0), \gamma'(0)^{\perp}$$

so we will try many configurations and compare the differences. We will discretize time such that $t_i \in \{0, ..., T\}$ such that $t_{i+1} = t_i + \Delta t$ where $\Delta t = 0.01$ and $T = 5$.

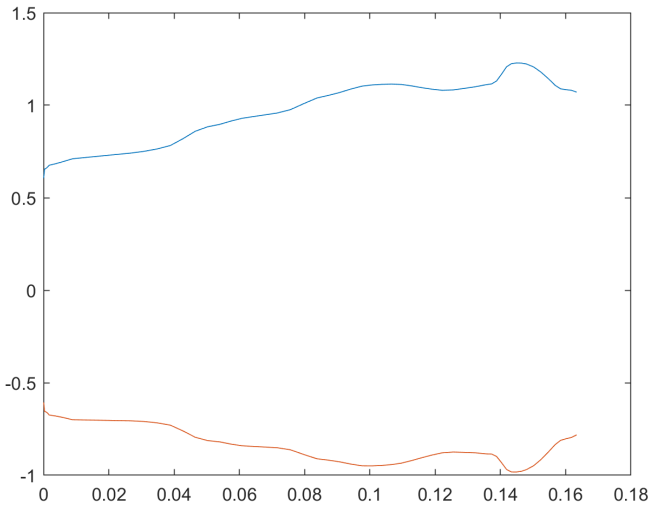
Let us first set $\theta = \mu = \sigma = 1, r(0) = 1, r'(0) = 0.5, \gamma(0) = 0, \gamma'(0) = 0.7$. We get the following random ribbon:



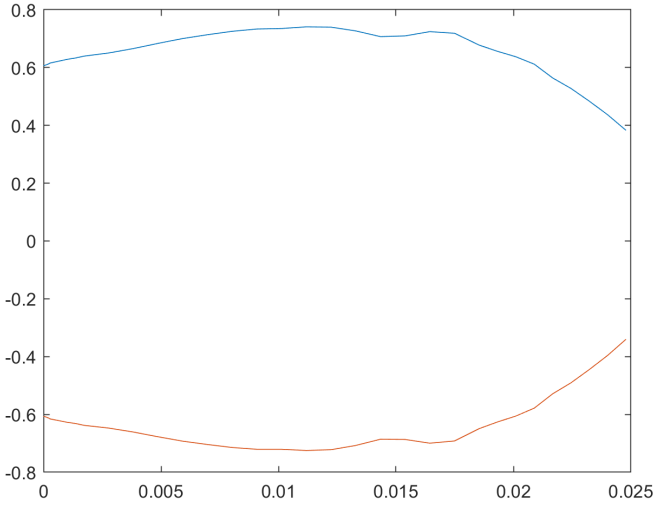
We see that the shape is relatively smooth. Let's try increasing σ to 5. We get the following:



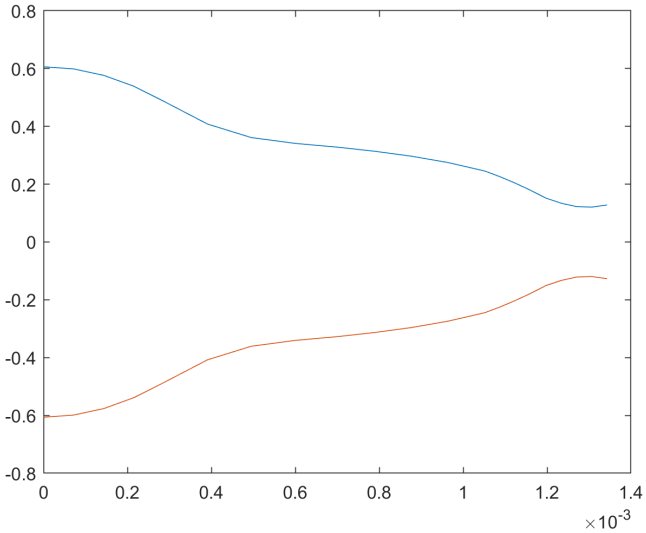
Increasing σ seems to increase the randomness of curvature. Let's keep all else the same and increase θ to 5. We get:



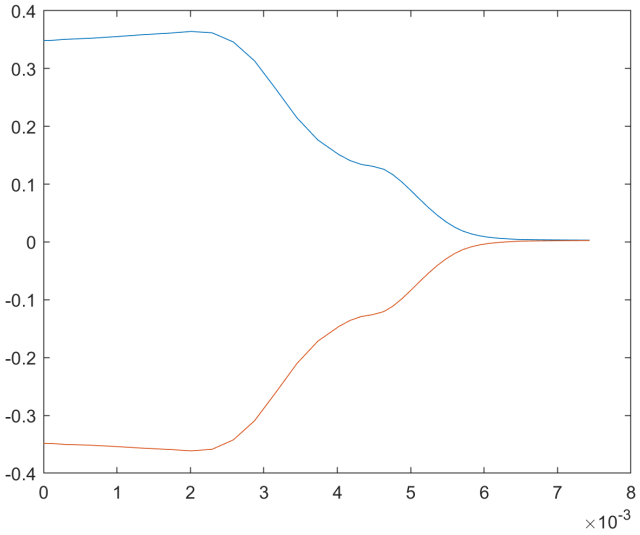
Let's now increase μ to 5 as well. We get:



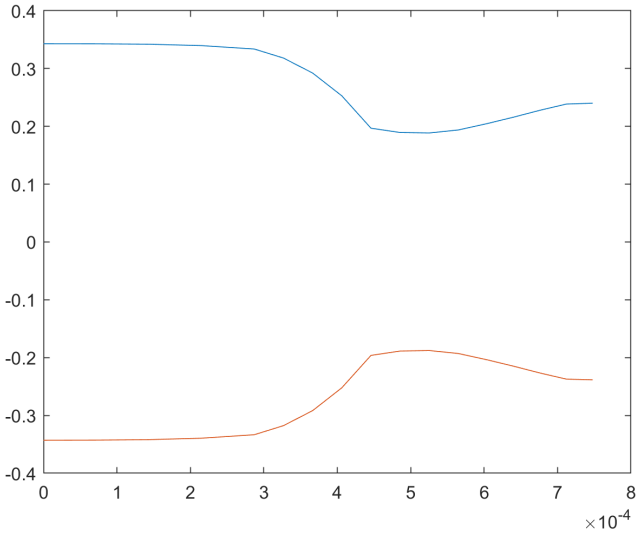
Now, let's try using a much larger σ, say $\sigma = 50$. Also, let's set $\theta = 10$ and leave $\mu = 5$. We get:



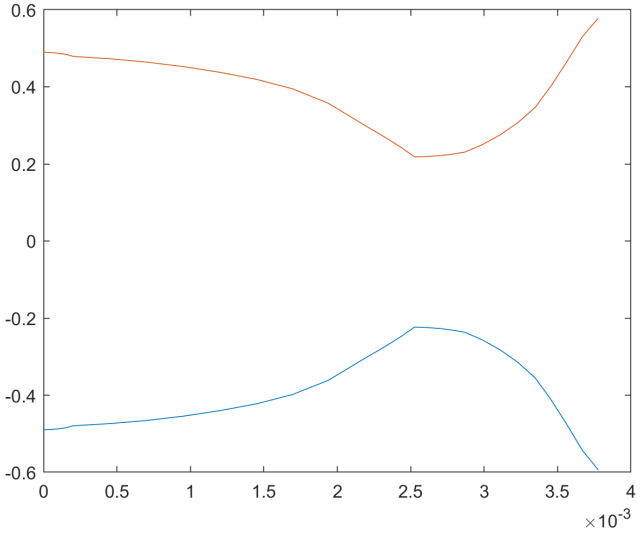
Now we try different initializations. Let's set $r(0) = 0.5$ and $r'(0) = 0.1$ leaving all else equal. We get:



Let's now try $r'(0) = -0.2$. We get:



Lastly, we change $\gamma'(0)^{\perp} = -1$. We also set $\sigma = 10$. We get:



Changing initializations and parameters do seem to affect the outcome of our simulation, but due to randomness, it is difficult to pinpoint exactly how. Certainly, setting $r'(0) > 0$ causes the shape to decrease radius initially and the opposite is true for $r'(0) < 0$. Changing σ clearly affects the volatility of the curve itself. Nonetheless, keeping parameters constant can yield completely different outcomes.

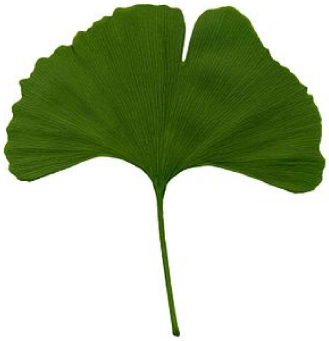
1.1 MATLAB Code

```
function [x] = sample_o_h(th,mu,sig,dt,T)
%SAMPLE_O_H samples an Ornstein-Uhlenbeck Process with given params
t = 0:dt:T;                 % Time vector
x = zeros(1,length(t)); % Allocate output vector, set initial condition
%rng(1);                     % Set random seed
for i = 1:length(t)-1
        x(i+1) = x(i)+th*(mu-x(i))*dt+sig*sqrt(dt)*randn;
end
end

function drdt = rODE(t, r, Xt,C)
C = interp1(Xt, C, t); % Interpolate the data set (Xt, C) at times t
drdt = [r(2); (1-C*sqrt(1-r(2)^2)-r(2)^2)/r(1)]; % Evalute ODE at times t
end

%[t,g] = ode45(@(t,g) curveODE(t,g,K,Xt), tspan, ic, opts)

function dgdt = curveODE(t,g,K,Xt)
K = interp1(Xt, K, t); % Interpolate the data set (Xt, K) at times t
dgdt = [g(2); K*(g(2))]; % Evalute ODE at times t
end

% Set initial Params
th = 10;
mu = 5;
sig = 10;
dt = 0.01;
T = 5;
Xt = 0:dt:T;

% Sample OH process
X1 = sample_o_h(th,mu,sig,dt,T);
X2 = sample_o_h(th,mu,sig,dt,T);
```

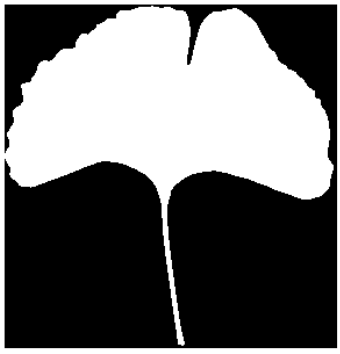
```
% Values necessary
C = 1/2*(exp(X1) + exp(X2));
Z = 1/2*(exp(X1) - exp(X2));

% Tspan and initial conditions for r
tspan = [0 T];
ic = [0.5;-0.2];
[~,r] = ode45(@(t,r) rODE(t,r,Xt,C), tspan, ic);

% Find where r stopped
m = length(r);
% Use new time discretization
dt2 = T/m;

% Find time where |r'(t)| < 1
% Same as where r'(t) becomes imaginary
n = sum(imag(r(:,2)) == 0);
r = r(1:n,:);
K = Z(1:n)./r(:,1)';

% Solve for gamma with new time discretization
Xt2 = 0:dt2:((n-1)*dt2);
tspan2 = [0 (n-1)*dt2];
ic2 = [0;-1];
[t,g] = ode45(@(t,g) curveODE(t,g,K,Xt2), tspan2, ic2);

% Use shortest length
if length(g) > length(r)
        n = length(r);
else
        n = length(g);
end

% Get each gamma
gt = g(1:n,1);
gp = diff(g(1:n,1));
gperp = g(1:n,2);
gp(length(gp)+1) = gp(length(gp));

% Get each r
rt = r(1:n,1);
rp = r(1:n,2);

% Solve for P and plot
Pp = gt + rt.*(-rp.*gp + sqrt(1 - rp.^2).*gperp);
Pn = gt + rt.*(-rp.*gp - sqrt(1 - rp.^2).*gperp);

% Get t
t_s = t(1:n);

plot(t_s,Pp, t_s, Pn);
```

2 Finding the Medial Axis of Leaves

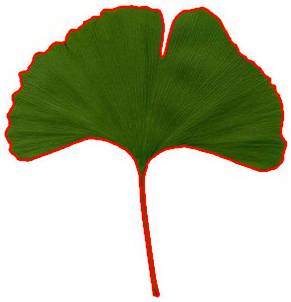
We seek to find the medial axis of an array of complex numbers representing the border of a leaf. To do so, we first acquire images of various different kinds of leaves using Google. First, take, for example, this leaf:



If we convert to black and white and take the complement (reverse black and white), we get this image:



Using MATLAB, we can easily detect the boundary of the above image, which, when plotted on top of the original image, looks like this:



Define the above shape as S and its boundary as ∂S. We can use the code provided in *Pattern Theory* to detect and plot the medial axis of S using our discretization of ∂S. So far, we have N Cartesian points (x_i, y_i), $i = 1, ..., N$ that make up the boundary of the shape. The first step is converting these coordinates to $\mathbb{C}$ such that $z_j = x_j + iy_j$, $j = 1, ..., N$. Each z_j together forms $\mathbb{S} = \{z_j\}$, the polygon representation of ∂S. Using the Voronoi decomposition and Delaunay triangulation, we can uncover its medial axis. Define

$$V(\mathbb{S}) = \{z \in \mathbb{C} | \ \exists i \neq j, ||z - z_i|| = ||z - z_j|| = d(z, \mathbb{S})\}$$

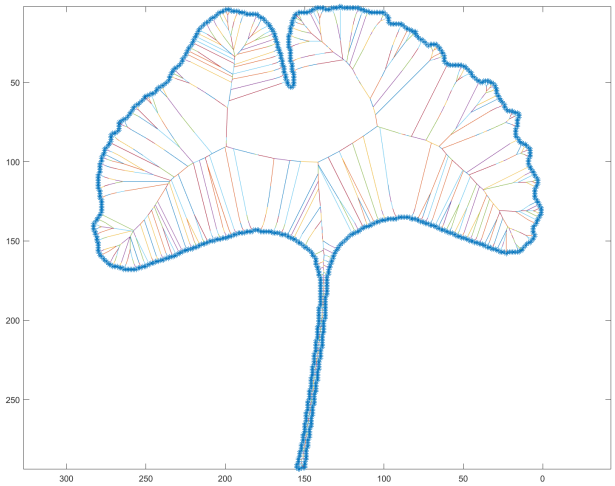
where

$$d(z, \mathbb{S}) = \min_i d(z, z_i).$$

Furthermore, let $h(\mathbb{S})$ be the set of points $z \in V(\mathbb{S})$ that are equidistant from adjacent points z_k, z_{k+1}. Then, we obtain the medial axis $Ax(S)$ as follows:

$$Ax(S) = \lim_{\mathbb{S} \to \partial S} [V(\mathbb{S}) - h(\mathbb{S})]$$

i.e. $Ax(S)$ is the set of centers for all maximal disks in S. If we apply the code from pattern theory, we achieve the following representation of the medial axis of S:



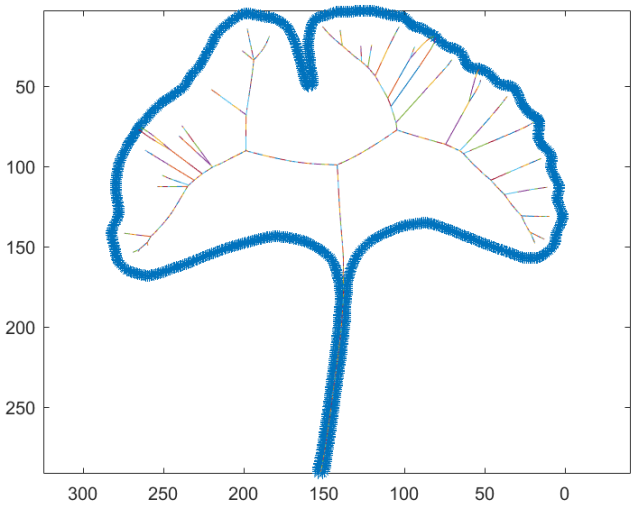
The problem with this representation of the medial axis is that it is overcomplicated due to the jagged nature of the boundary. To fix this, we can smooth the boundary using the geometric heat equation. We first represent ∂S as a closed, smooth curve dependent on time $C_t(s)$. The curve then evolves over time as follows:

$$\frac{\partial C_t(s)}{\partial t} = -\mathcal{K}_t(s)\vec{n}_t(s)$$

where $\mathcal{K}_t(s)$ is the curvature of $C_t(s)$ and $\vec{n}_t(s)$ are its normals. This evolution will make the length of the curve decrease: it pushes concave parts inside while pulling convex parts outside. In this sense, if we run the geometric heat equation for a short period of time, we will preserve the overall shape of ∂S while smoothing it. To do this in MATLAB, we implement the following finite difference method to approximate small time steps:

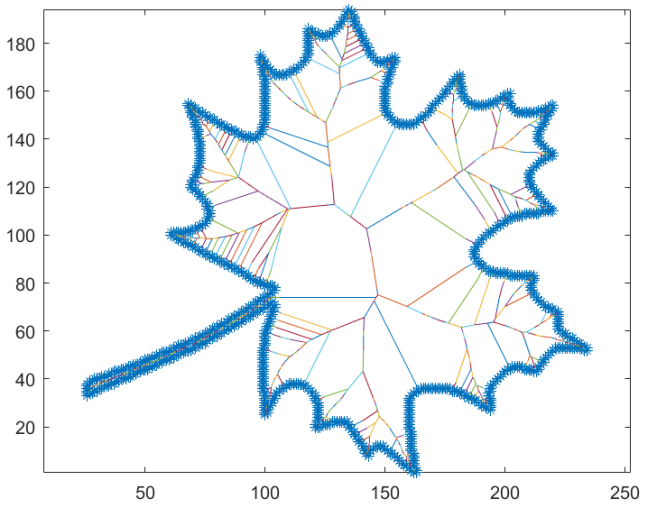
$$\frac{C(t+\Delta t, s) - C(t,s)}{\Delta t} = -\mathcal{K}_t(s)\vec{n}_t(s)$$
$$C(t+\Delta t, s) = C(t,s) - \Delta t \cdot \mathcal{K}_t(s)\vec{n}_t(s)$$

At each time step, we calculate the curvature and normals and update until we achieve the desired smooth boundary. To make sure we decrease the length of the curve, we remove points that would otherwise intersect. For instance, if we set $\Delta t = 0.05$ and run 12 iterations, we achieve the following boundary together with its medial axis:

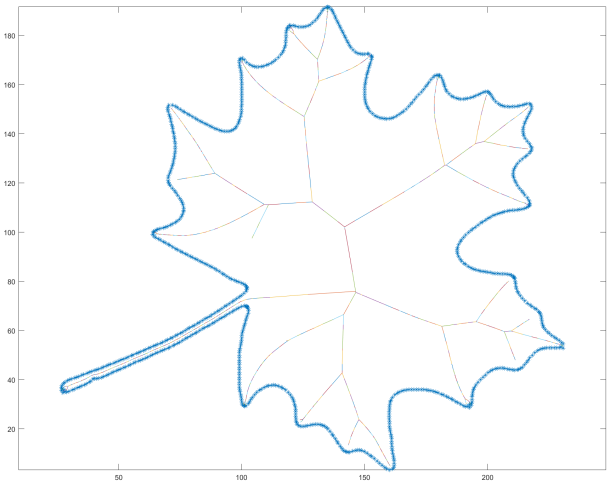


We see not only that the boundary is smoother, but the medial axis is more well-defined.

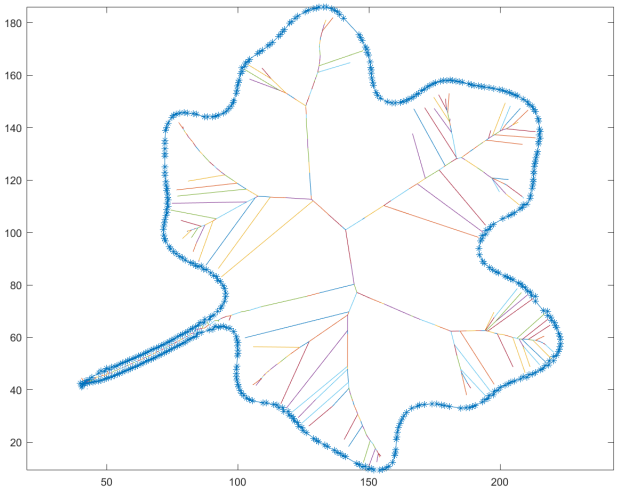
Let's consider the case of a "pointy" leaf, such as a maple leaf. If we compute the medial axis without smoothing, we get the following:



We see that the medial axis is overly complex. After smoothing with a time step of 0.02 for 50 iterations, we achieve the following:



We see that the points become rounded, giving way to a more well-defined medial axis. We must be careful in our choice of Δt and the duration of our geometric heat equation, so it is important to test different configurations. For instance, if we run for many more iterations, we start to "over-smooth:"



2.1 MATLAB Code

```
% Import leaf image
I = imread('leaf5.jpg');

% Convert to black and white image
BW = im2bw(I);
BW = imcomplement(BW);
BW = imfill(BW,'holes');

% Detect boundary
```

```
dim = size(BW);
col = round(dim(2)/2)-90;
row = min(find(BW(:,col)));
boundary = bwtraceboundary(BW,[row, col],'S');

% Plot leaf image with boundary
%{
imshow(BW)
hold on;
plot(boundary(:,2),boundary(:,1),'r','LineWidth',3);
%}
% Initial boundary
C = boundary;

% Smoothing using geometric heat equation
for dt = 0.05:0.05:0.5
        N = LineNormals2D(C);
        K = LineCurvature2D(C);
        d = dt*N.*K;
        % Delete points that would intersect
        % Gives 'decreased length' of curve
        [row,~] = find(abs(d) >= 1);
        C(row,:) = [];
        d(row,:) = [];
        C = C - d;
        % Clear isnan (non differentiable points)
        [row,col] = find(isnan(C));
        C(row,:) = [];
end

%{
% Plot smoothed curve and normals
plot(C(:,2),C(:,1),'g','LineWidth',2)
hold on;
%plot(boundary(:,2),boundary(:,1),'r','LineWidth',1)
plot([C(:,2) C(:,2)+10*N(:,2)]',[C(:,1) C(:,1)+10*N(:,1)]');
%}

% Medial Axis (re-order so plotting lines works)
z = C(:,2) + C(:,1)*1i;
C = [real(z),imag(z)];
z = C(:,1)' + C(:,2)'*1i;

% This code assumes a polygon is given by a vector z of complex numbers
plot(z,'-*'), hold on, axis equal
xmax = max(real(z)); xmin = min(real(z)); ymax = max(imag(z)); ymin = min(imag(z));
maxlen = max((xmax-xmin), (ymax-ymin))/20;

% Get Delaunay triangles and compute their circumcenters
tri = sort(delaunay(real(z),imag(z))')';
u = z(tri(:,1)); v = z(tri(:,2)); w = z(tri(:,3));
dot = (u-w).*conj(v-w);
m = (u+v+1i*(u-v).*real(dot)./imag(dot))/2;
r = abs(u-m);

% Prune the exterior triangles. m is now the vertices of the medial axis
inside = imag(dot) < 0;
triin = tri(inside,:);
m = m(inside); r = r(inside);
nt = size(triin,1);
```

```
% Find edges of the medial axis
B = sparse([1:nt 1:nt 1:nt], triin(:), ones(1,3*nt));
[a,b,c] = find(B*B'>1);
ind = a>b; a = a(ind); b = b(ind);
plot([m(a); m(b)], '+')
%plot(m(a));
%{
% Subdivide long segments in the medial axis
numsub = ceil(abs(m(a)-m(b))/maxlen);
ind = numsub > 1;
delta = 1./numsub(ind);
newm = m(a(ind))*(delta:delta:1-delta) + m(b(ind))*((1-delta):-delta:delta);
newr = abs(newm - z(dsearchn([real(z), imag(z)], tri, [real(newm), imag(newm)])));
m = [m newm]; r = [r newr];
nm = size(m,2);
plot(m,'*')
%}
```